

EL977166584US

**SOFTWARE-CONTROLLED CACHE SET MANAGEMENT**CROSS-REFERENCED APPLICATIONS

This application relates to co-pending U.S. patent applications entitled "PSEUDO LRU FOR A LOCKING CACHE" (Docket No. AUS920020476US1), "IMPLEMENTATION OF A PSEUDO-  
5 LRU ALGORITHM IN A PARTITIONED CACHE" (Docket No. AUS920020475US1), and "SOFTWARE-CONTROLLED CACHE SET MANAGEMENT WITH SOFTWARE-GENERATED CLASS IDENTIFIERS" (AUS920030531US1), all filed concurrently herewith.

10 TECHNICAL FIELD

The invention relates generally to cache management and, more particularly, to software-controlled cache set management.

15 BACKGROUND

Typically, caches are small, fast storage buffers employable to store information, such as instruction code or data, in order for a processing device to more quickly and efficiently have access to the information. Typically, it  
20 is faster for the processing device to read the smaller memory of the cache than to read a main memory. Also, with the rapid increase of intensive computational requirements, such as graphical processing and advanced numerical analysis, their importance in a computing system will only  
25 increase.

A cache is a limited resource. One way to allocate sets of a cache is to employ a hardware least recently used (LRU) function to determine replacement of sets. There are other hardware replacement algorithms including most recently used and first in first out. Given the LRU information, the cache determines the last set accessed with the cache in the case of a cache miss. The cache replaces the LRU information in the event of a cache miss, that is, the cache did not have the requested information. This replacement allows the processor to quickly access the selected new information the next time this information is selected. Furthermore, this replacement also increases the chances of the processor finding associated information, as the replaced set cache data is likely to have temporal or spatial locality.

However, there are issues related to the employment of a hardware LRU function in a cache. For instance, some information, such as streaming data, tends to replace large amounts of useful instructions or data in a cache. It would be desirable to isolate this replacement to certain sections of the cache, and leave other sections to be used opportunistically. Furthermore, there are certain critical data and instructions which will be replaced in the cache by the normal LRU. It would be desirable to lock these in the cache and not allow the LRU to replace them. Therefore, what is needed is a cache management scheme which overcomes these limitations.

#### SUMMARY

The present invention provides a software controlled data replacement for a cache. The software controlled data replacement employs a class identifier and a tag replacement

control indicia. The software controlled data replacement for a cache further provides a replacement management table, employable to read the class identifier to create the tag replacement control indicia. The cache comprises a plurality of sets. The cache is employable to disable a replacement of at least one of the plurality of sets as a function of the tag replacement control indicia.

#### BRIEF DESCRIPTION OF THE DRAWINGS

10 For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following Detailed Description taken in conjunction with the accompanying drawings, in which:

FIGURE 1 schematically depicts a cache set access system;

FIGURE 2 schematically depicts a replacement management table;

FIGURE 3 illustrates two rows of a replacement management table;

20 FIGURE 4 illustrates a range start register;

FIGURE 5A illustrates a range mask register;

FIGURE 5B illustrates one configuration of a classID register (CIDR);

FIGURE 6 schematically depicts a classID generator; and

25 FIGURE 7 illustrates a method for employing the cache subsystem for enabling and disabling a replacement of a set.

#### DETAILED DESCRIPTION

30 In the following discussion, numerous specific details are set forth to provide a thorough understanding of the present invention. However, those skilled in the art will

appreciate that the present invention may be practiced without such specific details. In other instances, well-known elements have been illustrated in schematic or block diagram form in order not to obscure the present invention  
5 in unnecessary detail. Additionally, for the most part, details concerning network communications, electro-magnetic signaling techniques, and the like, have been omitted inasmuch as such details are not considered necessary to obtain a complete understanding of the present invention,  
10 and are considered to be within the understanding of persons of ordinary skill in the relevant art.

It is further noted that, unless indicated otherwise, all functions described herein may be performed in either hardware or software, or some combination thereof. In a  
15 preferred embodiment, however, the functions are performed by a processor, such as a computer or an electronic data processor, in accordance with code, such as computer program code, software, and/or integrated circuits that are coded to perform such functions, unless indicated otherwise.

20 Referring to FIGURE 1, generally, a CPU with cache memory 100 employs software to manage a cache set replacement algorithm. The software allows the sets of an L2 cache 170 to be associated with different replacement strategies, implemented within a replacement management  
25 table (RMT) 160, for different requested memory address ranges.

In FIGURE 1, the reference numeral 100 generally designates a cache subsystem in which an L2 cache 170 is employed. The CPU with cache memory subsystem 100 comprises  
30 a central processing unit (CPU) 110, the coupled L2 cache 170 and a memory bus 180. In one embodiment, the CPU 110 comprises a reduced instruction set (RISC) IC processor.

The CPU 110 comprises an instruction address register 120 and a data address register 125. The instruction address register 120 is coupled to a level one (L1) instruction cache 130. The data address register 125 is coupled to an L1 data cache 135. The L1 instruction cache 130 is coupled to a first range register 140. The L1 cache 135 is coupled to a second range register 145. A MUX 147 is coupled to both the registers 140, 145 and further coupled to an RMT 160, which is coupled to the L2 cache 170.

10 The instruction address register 120 is coupled to the L2 address register 150 through branch predictor 126. The data address register 125 is coupled to the L1 data cache 135. The data cache 135 is, in turn, coupled to the L2 address register 150. The L2 address register 150 is coupled to the L2 cache 170. The L2 cache 170 is divided into a plurality of sets. Each set is correlated to at least one address range within the RMT 160. If the requested information is not in the L2 cache 170, these sets are updated with the requested information. The sets are updated according to whether the sets are to be replaced or not, as defined by the RMT 160. The determination of which set is enabled or disabled for a requested memory address comprises a function of the requested address. Generally, in the CPU with cache memory subsystem 100, an LRU function employs selection among sets for a given corresponding classID for any plurality of sets allowed for replacement by the RMT 160. Although, in FIGURE 1, the L1 caches 130, 135 and the L2 cache 170 are illustrated, those of skill in the art understand that other caches, or other temporary information storage devices, can also be employed.

Generally, in the CPU 110, the instruction address register 120 commands an L1 instruction cache 130 to

retrieve instructions from the memory of the L1 instruction cache 130. In the CPU with cache memory subsystem 100, the instruction address register 120 first checks the tag of the L1 instruction cache 130 for an information request. If the requested information, corresponding to a specific address, is stored in the L1 instruction cache 130 according to the tag of the L1 instruction cache 130, the requested information is then placed in an instruction buffer 127, decoded, and issued as an instruction to execution units.

10 In one embodiment, a branch predictor 126 is also employed. Typically, the branch predictor 126 makes predictions as to the outcome of a branch instruction. In FIGURE 1, the floating point register (FPR) 161 is coupled to a floating point unit (FPU) 165, and the register 162 is coupled to a fixed point unit (FXU) 166 and a load store unit (LSU) 176.

15 The registers 161, 162 and the LSU 176 are coupled to an L1 data cache 135.

In the event of a hit in the L2 cache 170, the hit information is forwarded to the requesting L1 instruction cache 130 or the L1 data cache 135. From the caches 130, 135, the information is accessible by the CPU (not shown) of the CPU 110.

20

However, if the tag of the L1 instruction cache 130 indicates that the desired information is not stored in the L1 instruction cache 130, the instruction address register 120 sends the requested address to the L2 address register 150 to determine whether the requested information is in the L2 cache 170 instead. Furthermore, if the tag information of L1 instruction cache 130 indicates that the desired information is not stored in the L1 instruction cache 130, the requested address is sent to the range register 140. Similarly, a miss in the L1 data cache 135 sends the

25

30

corresponding address to the range register 145 and a request to the L2 address register 150. In the event of a miss of the L1 data address register 125, the desired data address originally derived from the instruction address register 120 is transmitted to the data range register 145 from the L1 data cache 135. Generally, the data address register 125 is then multiplexed with the output of the instruction address register 120, in the L2 address register 150, in the event of a miss of the L1 data cache 135.

10 In the range registers 140 and 145, the address that was not found in the L1 instruction cache 130 or L1 data cache 135, respectively, is tested to determine whether the selected address falls within a range of memory addresses. If it does not, a "default" class identifier (classID) is generated. However, if the selected address does fall within the range of memory addresses associated with the range registers 140, 145, a classID is created corresponding to the appropriate address range, such as classID 1, classID 2, and so on. The classID is transmitted through a MUX 147 to the RMT 160.

25 Generally, a class identifier is employed as an index to a row of the RMT 160. The RMT 160 generally determines whether a given set of the L2 cache 170 can be replaced or not. The status information of a row of the classID is then converted to an L2 tag replacement control indicia. The RMT 160 transmits the L2 tag replacement control indicia to the L2 cache 170.

Typically, the L2 address register 150 checks the tag of the L2 cache 170 to determine if the desired information is stored in the L2 cache 170. If the desired information is found in the L2 cache 170, the information is retrieved

from the L2 cache 170 and the L2 tag replacement control indicia is employed to update the LRU.

However, if the selected address is not found in the L2 cache 170, the corresponding requested information is  
5 retrieved from a main memory, such as through the memory bus 180. This retrieved information is then stored in a set of the L2 cache 170. However, the information that is stored in the L2 cache 170 is stored in a set of the L2 cache 170 as a function of the received L2 tag. For instance,  
10 addresses falling within the classID of "one" could be targeted as always stored in set four of the cache 170, depending upon the particular software configuration of the RMT 160. Addresses falling within classID of "three" and "four" could be both selected to overwrite the same set in  
15 the L2 cache 170, the cache set seven. A classID of "zero" could require that all sets of the L2 cache 170 not replace sets zero through two. These directives would be conveyed to the L2 cache 170 by the L2 tag replacement control indicia. These directives could also be modified by  
20 software, so that different sets eligible for replacement are defined for different given class IDs.

Turning now to FIGURE 2, schematically illustrated is a replacement management table 160. The table 160 comprises a matrix of classIDs, generated by the registers 140, 145  
25 crossed with the defined sets of the L2 cache 170. Generally, the table 160 illustrates the decisions made by software of the operating system executing on the CPU with cache memory subsystem 100 as to the replacement management status of the sets within the L2 cache 170, as they  
30 correlate to a given classID.

In FIGURE 2, the RMT 160 is a software managed table. Software maintains the structure and interprets the meaning



of the RMT 160 entries which specify which cache sets are to be used for specific ClassIDs. In a further embodiment, no hardware checks of the RMT 160 for accuracy are made. Generally, the RMT 160 is employed in the mapping of a missed address range to a set or sets of the L2 cache 170. In one embodiment, the address range associated with a classID is an effective address range. In another embodiment, the address range associated with a classID could be a real or physical address. In the registers 140, 145, the missed address is mapped to a classID. Generally, the classID is associated with one or more memory regions or given address range. In one embodiment, classID zero corresponds to any address range that is not specifically provided for by the other classIDs. The given classID is then transmitted to and employed as an index to the RMT 160. Using the classID as an index, the RMT 160 is accessed, the replacement information is read, and the tag replacement control indicia is generated.

In FIGURE 2, the RMT 160 is a matrix of eight by eight. In other words, the L2 cache 170 is 8-way set associative (that is, it has 8 sets). Therefore, each RMT 160 entry has 8 bits. However, those of skill in the art understand that other RMT sizes are within the scope of the present invention. In FIG. 2, the RMT 160 has "1"s defined in classID row zero, from sets 0 through 2, and "0"s in the rest of the row. Therefore, if the data is to be replaced in the L2 cache, it can be replaced in the first three sets, sets 0 through 2, of the L2 cache for memory accesses associated with classID zero. Furthermore, the classID zero has exclusive use of these sets. Therefore, sets 0 through 2 of the L2 cache are exclusive to the classID zero. In one embodiment, classID zero corresponds to an address range

having no specified classID. That is, an address miss that is not specifically provided for in the range registers 140, 145 is given a default classID zero.

For classID one, the RMT 160 has a "0" defined in sets  
5 0-2 and sets 4-7, and a "1" for set 3. Any data corresponding to classID one is not to be placed in sets 0-2 and sets 4-7. Instead, the data is placed in set 3. ClassID 2 and classID 3 both replace the same set, set 7. Therefore, both classID 2 and classID 3 use set 7. ClassID  
10 4 has a plurality of sets that are valid candidates for replacement. These are sets 4, 5 and 6.

In the illustrated embodiments, classIDs 5-8 are not used. That is, all entries in each classID is "0". However, those of skill in the art understand that the  
15 logical set and classID determinations expressed as "1"s and "0"s, as shown in FIGURE 2, are for purposes of illustration, and that other logical set and classID determinations are within the scope of the present invention, such as through software.

20 This replacement of information within the allowed sets is performed by the LRU function as indicated by the RMT 160. Similarly, information corresponding to any other classID is replaced in the various sets of the L2 cache 170 according to the software managed RMT table 160 and the LRU  
25 function.

Turning now to FIGURE 3, schematically illustrated are two rows of information of the RMT 160. In FIG. 3, an "a" bit, or the "algorithm" bit, is disclosed. This allows the subsystem 100 to choose an algorithm other than the LRU for  
30 the replacement of eligible sets. In one embodiment, the algorithm is a most recently used algorithm or a first in first out algorithm. The rows have a field corresponding to

the classID and a field "set enable bits" which enable or disable sets of the L2 cache 170 for a given classID. For instance, set enable bits for RMT\_index 1 could be "0,1,0,1,1,0,0,0." The next field, the valid bit "v" field, indicates that the given RMT row contains valid information. The bypass bit "b" indicates whether the operation should be cached at the level of the cache, such as an L1 cache, an L2 cache, and so on. The algorithm bit "a" specifies the replacement algorithm to be used for this class.

Accessing an invalid RMT row, according to the "v" valid bit entry, typically results in a default class employed by the RMT 160. In one embodiment, the default class is ClassID zero. In a further embodiment, accessing an invalid RMT row generates an interrupt to the CPU. In one embodiment, all bits are set for enable for an invalid ClassID of the RMT 160. In a further embodiment, an error signal is returned to the operating system instead of generating an interrupt. This occurs if the process running on the CPU, but not the operating system itself, are terminated.

In a further embodiment, setting the bypass "b" bit indicates that data of a certain classID is not to be cached at this hierarchy level, such as an L1 cache, an L2 cache, and so on. For data corresponding to this classID, this data should be passed to the bus serving the CPU directly. In one embodiment, the data that is to be passed directly to the bus also exists in this level of the hierarchy.

Turning now to FIGURE 4, illustrated is a range start register (RSR) 400. Generally, a requested missed operand address is masked with a range mask register (RMR) and then compared to the RSR. The output of the masking of the RMR is then tested to determine whether there has been a range

hit, as determined by a comparison to the RSR. This comparison typically occurs within the range registers 140, 145. The RMR is the mask that defines the ending address range.

5 In one embodiment, the address range registers 140, 145 are accessible with a "move to/from special purpose register" instruction. Permission to access the selected address ranges at these registers 140, 145 is granted only in a "privileged" state of the operating system. In a  
10 further embodiment, the "move to/from special purpose register" instruction is issued in a PowerPC environment.

Turning now to FIGURE 5A, illustrated is a range mask register (RMR) 500. Generally, an address range is tested by the range registers 140, 145 through employment of the  
15 mask qualities of the RMR and then performing a comparison with the RSR.

Generally, the RSR defines the starting address and the RMR defines the ending address that is to be tested for within the range registers 140, 145. This is performed by  
20 the RMR 500 masking the bits of a missed operand's address, and then comparing the missed operand to the bits of the RSR 400. In FIGURE 5A, the upper bits of the RSR 400 pass the starting address of the range to be tested, the lower bits, and the classID. Furthermore, the RMR masking operand does  
25 not mask the address range mode (that is, Real or Virtual) bit and the valid or disabled bit.

In the CPU with cache memory subsystem 100, the address register 140 employs the RSR and the RMR to map an effective address (EA) or a real address (RA). One example is a  
30 mapping for the CPU to enable load/stores and instructions retrieval. Typically, the size of the range to be tested is a power of 2 and the starting address of the range is a

range size boundary. In other words, in FIGURE 5A, the upper 32 bits of the operand's address and result of a bit-wise AND of the lower 32 bits of the operand's address with the RSR 400 create the upper address boundary to be tested.

5       Turning now to FIGURE 5B, illustrated is a classID register (CIDR). ClassID information passes through this register when the operand comes within the selected address range. Thereafter, the classID is readable. In one embodiment, the size of the classID is implementation  
10       specific.

Turning now to FIGURE 6, illustrated is a classID generator 600. The operand's address is received through input line 610 and an AND 630 performed upon it and the RMR mask 620 for bits 32-51. Then, bits 0-31 are imported from  
15       the requested missed address, and bits 32-51 are appended to bits 0-31. These are, in turn, compared to the RSR 400 in a comparator 650. This comparison is conveyed to an AND 660. The AND 660 receives the equality compare from CMP 650 and the V output of the RSR 640. The V output corresponds to  
20       the "valid" bit, and the R output corresponds to the "real" bit.

The output of the AND 660 corresponds to expressing a range hit. If the range hit was positive, then the classID, derived from employment of the CIDR mask 685, is generated.  
25       This indicia of a range hit is passed to the OR 695 gate as the classID corresponding to the particular range. These steps are repeated for each address range in the registers 140, 145.

Turning now to FIGURE 7, illustrated is a method 700  
30       for setting the enabling and disabling of replacement for the sets of the L2 cache 170. In FIG. 7, the L2 cache 170 comprises a plurality of sets. Although steps of the method

700 are described with regards to the employment of the instruction address register 120, the L1 instruction cache 130 and the range register 140, similar steps are performed for the employment of the data address register 125, the data cache 135 and the range register 145.

In step 710, the CPU requests information corresponding to a particular address. In step 720, the instruction address register 120 queries the tag of the L1 cache 130 if there is information corresponding to the particular address requested stored in the L1 instruction cache 130. If there is, the requested information is conveyed to the CPU 110 and the method 700 stops.

However, if there is a miss of the L1 instruction cache 130, then step 729 determines if the requested memory address is defined in the range register 140. In step 730, if the memory address is defined within a memory range, a classID is generated corresponding to the range to which it corresponds. If the range is not defined, then in step 740 a default classID is generated.

In step 750, the classID is read by the RMT 160 and is used as an index for the corresponding row in the RMT 160. For instance, in FIGURE 3, classID two corresponds to "1,0,1,1,1,0,0,0." In other words, the software defines that RMT 160 sets 0, 2, 3, and 4 are eligible for replacement and sets 5, 6 and 7 are not eligible for replacement.

In step 755, tag replacement control indicia is created by the RMT 160. Generally, the tag replacement control indicia is employed to control the replacement eligibility of sets of the L2 cache 170. In step 760, the requested address is then requested from the L2 cache 170. In step 770, if the L2 cache 170 has the information corresponding

to the requested address, then that information is forwarded to the CPU 110 and the method 700 ends in step 775.

However, if there is a "miss" of the requested data address in step 770 (that is, information corresponding to the requested data address is not stored in the L2 cache 170), then replacement eligibility of at least one set of the L2 cache 170 is configured in step 780. In other words, at least one set of the L2 cache 170 is configured as eligible for replacement or configured as not eligible for replacement as a function of the L2 tag replacement control indicia. In one embodiment, all sets of the L2 cache 170 are configured as enabled or disabled for a given L2 tag replacement control indicia.

In step 790, the method 700 overwrites information in a set of the L2 cache 170 that is eligible for replacement. A set that is disabled does not get overwritten, however. Typically, the choice among those sets employable to be overwritten that is actually overwritten is a function of an LRU function.

It is understood that the present invention can take many forms and embodiments. Accordingly, several variations may be made in the foregoing without departing from the spirit or the scope of the invention. The capabilities outlined herein allow for the possibility of a variety of programming models. This disclosure should not be read as preferring any particular programming model, but is instead directed to the underlying mechanisms on which these programming models can be built.

Having thus described the present invention by reference to certain of its preferred embodiments, it is noted that the embodiments disclosed are illustrative rather than limiting in nature and that a wide range of variations,

modifications, changes, and substitutions are contemplated in the foregoing disclosure and, in some instances, some features of the present invention may be employed without a corresponding use of the other features. Many such  
5 variations and modifications may be considered obvious and desirable by those skilled in the art based upon a review of the foregoing description of preferred embodiments. Accordingly, it is appropriate that the appended claims be construed broadly and in a manner consistent with the scope  
10 of the invention.